Interested in learning more about security?

SANS Institute InfoSec Reading Room

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

Digging Deeper Into TCP/IP

This paper takes a close look at TCP/IP as a reference for the security professional. Its goal is to consolidate information from numerous sources into one place and to go deeper into the client/server relationship. It explores basic TCP/IP concepts such as encapsulation, IP and TCP definitions, client/server connections, and discusses less well-known concepts such as state. It concludes with an illustration of an entire TCP session (Figure 10), including absolute sequence and acknowledgement numbers, state identificat...

Copyright SANS Institute

Author Retains Full Rights

AD

Digging Deeper Into TCP/IP

Leah Wilson

GSEC Practical Version 1.2f

Abstract

This paper takes a close look at TCP/IP as a reference for the security professional. Its goal is to consolidate information from numerous sources into one place and to go deeper into the client/server relationship. It explores basic TCP/IP concepts such as encapsulation, IP and TCP definitions, client/server connections, and discusses less well-known concepts such as state. It concludes with an illustration of an entire TCP session (Figure 10), including absolute sequence and acknowledgement numbers, state identification, and client and server responses.

TCP/IP is an immense subject and this paper is by no means exhaustive, but references are supplied to help the reader further explore this topic.

Encapsulation

IP is the protocol that transports packets created by the TCP protocol. This TCP/IP combination allows users to do a variety of things with their computers, such as finger, telnet, FTP, and HTTP. The TCP/IP combo allows two computers to connect to each other and exchange information in a reliable way.

TCP packets are encapsulated by IP as part of being sent across a network. David Hoelzer's "TCP/IP Primer" (p. 6-7) gives an extremely easy-to-understand explanation of encapsulation. Delivering packets over a network is like delivering the mail. The media for delivering the mail is a combination of letters, envelopes, mailboxes, mail bags, mail trucks, etc. The media for delivering packets is TCP/IP. TCP packets contain data (letter) that is encapsulated by IP (envelope), which is further encapsulated by the Link layer, e.g. in most cases Ethernet frame headers (mailbox). Data is put out on the wire (post offices, sorting centers) and forwarded via routers (mail trucks, planes) to its final destination (receiver's mailbox), where this process is reversed – the frame header is removed (the letter is removed from mailbox), the IP header is stripped off (the envelope is opened), and the data is processed by the end application (the letter is read).

IP

IP (Internet Protocol) was defined in RFC 791 (ISI-IP) as an Internet Layer protocol whose purpose is to get data from a source to a destination. It does this by IP routing. A TCP packet is crafted on a network and sent out on the wire. IP encapsulates it, and uses the subnet mask to determine if the packet is destined for a host on the local network (Stevens, p. 44). If it is, IP uses Address Resolution Protocol (ARP) to look up the Media Access Control (MAC) address of the destination host then sends the packet directly to that machine. If the packet is destined for a host outside the local network, IP sends the packet to the next host. This host performs the same function of checking if the packet belongs to its network and delivering it if it does or forwarding it if it doesn't. (Stevens, p. 54-56). Routing schemes such as split horizon and split horizon with

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

poison reverse are generally in place to ensure that the next hop is closer to the destination, not further (Hedrick, section 2.2.1). Each time the IP packet passes through a hop, the Time-To-Live (TTL) field in the header is decremented by one. If this field reaches 0 and the packet is still not at its final destination, it is dropped and an Internet Control Message Protocol (ICMP) error message is sent back to the sender.

As part of getting data from point A to point B across the Internet, IP has knowledge of how much data the next hop can accept. IP looks at the Maximum Transmission Unit (MTU) of the next hop and determines if the packet can be accepted. If the packet is too big for the next hop, IP will break it down, or fragment it, into several smaller pieces, each of which is encapsulated in its own IP header. These new, smaller packets are then routed separately. To keep track of this, IP uses the 16-bit identification field, the 3-bit flags field, and the 13-bit fragment offset field in the IP header. When a packet is fragmented, all the fragments are given the same 16-bit identification field in their header. The last of the 3-bit flags field is used as the "more to come" bit – if it is set, there is another fragment to follow this one. The 13-bit fragment offset field contains the offset of this fragment from the original packet, and the total length field is adjusted to be the size of the fragment. A good description of fragmentation can be found in Northcutt (p. 36-41) and Stevens (p. 148-149). The point is that IP uses fields in the IP header to reassemble the fragments at the final destination.

Note that the two other bits in the 3-bit flags field of the IP header do have meaning. The first bit is reserved and always set to 0. The middle bit is designated the "do not fragment" bit – if it is set, IP will not fragment the packet. If the packet's size exceeds the MTU of the next hop and the "do not fragment" bit is set, the packet will be dropped and IP will generate an ICMP error message (Stevens, p.149, and ISI-IP, p. 25).

There is no concept of an IP "connection". IP will send packets from the same application over any number of routes. It has no knowledge if these packets are related to each other in any way, and it really doesn't care if the packet gets to its final destination or not. If a packet gets to the point where it cannot be further delivered, say it cannot find the specified host on the local network or the TTL in the IP header reaches 0, then IP magnanimously sends an ICMP message to the last host that sent the packet, but it does nothing further and it does not attempt to resend the packet.

So why is TCP/IP touted as reliable and connection-oriented if the IP part of it is unreliable and connectionless? That is because the state of the connection is managed by TCP. IP moves the data from one computer to another, one hop at a time. It doesn't care if the packet is TCP or UDP, or if its part of a bitmap image or part of a text document. All IP knows is how to get the packet to its destination. TCP's job is to know what to do with the packets IP delivers – it knows source and destination ports, it knows what order the data is supposed to be in, it knows when data doesn't arrive and it knows when to resend that data. TCP is the manager that understands the bigger picture.

TCP

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

The Transmission Control Protocol "provides a connection-oriented, reliable, byte stream service" (Stevens, p. 223). TCP was defined in RFC 793 (ISI-TCP), and is a Transport Layer protocol that manages the end-to-end communications between hosts.

TCP treats data from an application as a contiguous stream of bytes and groups bytes of data together into packets (Rodriguez, p. 207). It puts a TCP header on every packet, giving such information as the source and destination ports and the 32-bit sequence and acknowledgement numbers. These last two fields are the heart of the connection-oriented nature of TCP, and are discussed further in the section below.

TCP features error recovery, which means if a packet does not reach its destination, TCP resends it. Every time application data is put into TCP packets, a timer is set. If TCP does not receive an acknowledgement from the other side of the connection that the packet was received before the timer expires, it resends the packet. Note that TCP resends the entire TCP packet. If IP fragments a large TCP packet and all but one arrive at the destination, TCP cannot and does not send only the fragment that didn't arrive at the destination – it resends the entire TCP packet, which may be fragmented differently on the retransmission during IP routing (Stevens, 149).

Flow control through the windowing mechanism is another feature of TCP. Each side of the connection transmits to the other side how much data it is willing to accept at a time, so that one side will not send more data than the other side can handle. Flow control is not further discussed in this paper, but good explanations of it can be found in Stevens (p. 280-282) and Rodriguez (p. 208-212).

Now that a general understanding of how packets are transported has been established, the focus moves to looking more deeply into how a TCP connection is established.

How do they work together?

The postal example of encapsulation applies more to the User Datagram Protocol (UDP) than TCP. UDP packets are sent out but there is never any confirmation that they reach their final destination. The sender hopes that the packets arrive but is willing to tolerate some data loss if they don't. TCP is more reliable than UDP, in that it incorporates the idea of a connection to ensure that packets arrive properly. A better analogy would be to compare TCP to a telephone call. The caller (sender) knows when call gets through (the connection is made) and redials (retransmits) if it does not go through.

Packets may arrive out of order and may travel various different routes over the Internet but because of TCP sequence and acknowledgement numbers, the receiver of the packets knows how to properly reassemble them when they do arrive.

Connection-Oriented

What exactly does "connection-oriented" mean? It means two computers must establish a connection to each other before they can exchange data (Stevens, p. 229). Who is establishing this connection? How is it maintained? How is it ended?

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

It may help the reader to refer to Figure 10 while reading the following sections. This figure is based on an actual TCP connection to a Simple Mail Transfer Protocol (SMTP) server. The sequence and acknowledgement numbers in the figure were derived from hex output generated by snort. Snort is a powerful and free intrusion detection system and sniffer. More information about snort can be found at http://www.snort.org.

A connection is generally described in terms of a client-server relationship. A server computer has information that it is willing to "serve" to other computers. It contains some software application that implements TCP/IP, so connections and data requests from other computers, called clients, can be handled. In order for the client to ask the server for information, it needs to establish a connection. So the client crafts a TCP packet with the SYN flag set and sends it to the server. The server replies by sending back a packet with the SYN and ACK flags set, acknowledging that it received the client's SYN and initiating its own connection to the client via use of its own SYN. The client then responds with another ACK to acknowledge receipt of the server's SYN/ACK, and the famous three-way handshake is complete – a connection is established. The connection is called "full-duplex", meaning data is flowing from both the client side and the server side – it can travel in both directions (Stevens, p. 233).

Sequence and Acknowledgement Numbers

To keep track of the connection, sequence (SEQ) and acknowledgement (ack) numbers stored in the TCP header are used. When a connection is started, the client sends a packet with an initial 32-bit SEQ number that ranges from 0 to 4,294,967,295 (this is $2^{32} - 1$). According to RFC 793 this is supposed to be a random number with a seed based on a 32-bit clock with a low order bit incremented every 4 microseconds. Note that some implementations, such as some Berkeley-derived TCP implementations, violate the RFC and base this number on a low order bit incremented every 8 microseconds (Stevens, p. 232-233). Generating an appropriate initial sequence number is a difficult task, as acknowledged by Cisco in February 2001 when they released a software patch to fix a bug in their router's operating system that allowed the initial sequence number to be predicted. Improper implementations can allow hackers to predict sequence numbers and launch session hijacking attacks or close existing sessions. For more information on the dangers of improperly generated initial sequence numbers, refer to Noel's Davis' article "Predictable Initial Sequence Numbers".

Acknowledgement (ack) numbers give the next expected sequence number. This will be discussed more in later sections. The important point is that SEQ and ack numbers are the glue that holds the TCP connection together, and they also allow TCP to properly reassemble packets when they arrive at their destination.

Three-Way Handshake

In the three-way handshake, the client sends a SYN packet with an initial 32-bit sequence number and with an acknowledgement number (ack) of 0. As an example, taken from Figure 10, this is represented by:

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

Figure 1

Step 1:
Client ⟶ SYN (SEQ: 1023645922) a:0 ⟶ Server

"Absolute" numbers are used for all examples in this paper. These are the actual numbers (converted to decimal) that were found in the hex output from snort (which uses tcpdump) from the TCP header's SEQ and ack fields. The tcpdump program defaults to using "relative" sequence numbers that are normalized, and this convention is followed in many textbooks. These smaller numbers are easier to read, but absolute sequence numbers are used in this paper to prevent confusion converting back and forth from relative/normalized to absolute. The "-r" option in tcpdump can be used to force a display of absolute numbers. For more information on tcpdump, see http://www.tcpdump.org.

If the server is listening, it responds back to the client with a packet having the SYN and ACK flags set, and containing its own initial SEQ and an ack equal to the server's SEQ plus one. (If the server is NOT listening, the client will get back a message that the connection failed.) Essentially the server is saying "I ACKnowledge the SEQ you sent me and here's a SYN of my own to open my side of the connection". This looks like:

Figure 2

Step 2:
Client ⟵ SYN/ACK (SEQ: 2234678799) a:1023645923 ⟵ Server

Here's **my** initial SEQ!

I see your SEQ and raise you one!

This is telling the server that the next packet it sends had better have a SEQ = 1023645923. And indeed it does, so the client sends back a packet with just the ACK flag set, and with updated SEQ and ack numbers.

Figure 3

Step 3:
Client ⟶ ACK (SEQ: 1023645923) a:2234678800 ⟶ Server

Here's **my** SEQ!

I see your SEQ and raise you one!

The three-way handshake is how a typical TCP connection is started. Notice the relationship of SEQ and ack numbers:

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

Figure 4

Step	SEQ #	ack #	SYN	ACK
1	X	0	1	0
2	Y	X+nBytes	1	1
3	X+nBytes	Y+nBytes	0	1

"nBytes" refers to the number of bytes sent in the packet's payload.

The ack number is the next expected sequence number that one side of a connection expects. Stevens defines it as "the sequence number plus 1 of the last successfully received byte of data" (pg. 226). During the three-way handshake, since there is no connection yet, no data is yet being sent (ack = 0), so nBytes = ack + 1 = 0 + 1 = 1. A packet will have an ack number only if the ACK flag is set.

Data Exchange

Once the connection has been established, the client and server can exchange data. For a telnet session, this data might be commands issued to a remote computer, say a user logging onto his work mainframe from home. For HTTP, this could be a GIAC student connecting to a web site then using FTP (another TCP connection!) to download a primer that explains the basics of TCP/IP. For SMTP, this could be a user sending email (the application translates the user's actions and data into SMTP commands that it sends to the server) or a hacker telnet-ing to port 25 and trying to verify user IDs.

When data is sent over a connection, the PSH flag in the TCP header gets involved. In most Berkeley-derived implementations, the PSH flag is set by TCP to signal there is no more data to follow, i.e. the packet with the PSH empties the buffer of what is being sent (Stevens, p.284). This differs slightly from the original RFC 793 (IIS-TCP, section 3.8).

Payload

SEQ and ack numbers are affected by the payload packet size. In Step 5 of Figure 10, notice the client is sending a "help" command to the SMTP server. It has the ACK and PSH flags set as in Figure 5:

Figure 5

Step 5:
Client ⟶ ACK/PSH (SEQ: **1023645923**) a:2234678893 ⟶ Server

"help"

The server's response is to send help information to the user (for SMTP, this is a list of valid SMTP commands). It sends this data in a packet (Step 7), also with an ACK/PSH:

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

Figure 6

Step 7:
Client ← ACK/PSH (SEQ: 223478893) a:**1023645928** ← Server

A list of SMTP commands

Notice the numbers in bold. Step 7's ack is not one more than the last SEQ it received, it is five more than Step 5's SEQ. That is because the packet in Step 5 sent the four characters "h", "e", "l", "p". The server is acknowledging to the client that it received the four bytes of data by setting its ack to the last successfully received byte (1023645923 + 4) = 1023645927, plus one to equal 1023645928. This jump in ack numbers happens every time a packet contains payload.

The analyst will rarely have to go into this much nitty-gritty detail about payload size. It is much easier and faster to use tcpdump with relative numbers, which also provides a start and end byte range of the data and a byte count.

Notice Step 6:

Figure 7

Step 6:
Client → ACK (SEQ: 1023645923) a:2234678893 → Server

This has the exact same SEQ and ack as Step 5, but only the ACK flag is set and there is no payload. In fact, the whole connection has extra ACKs sprinkled liberally throughout. These extra ACKs are the application's way of continually verifying that the connection is still up and running, and their use varies from implementation to implementation.

Shutdown

There are two ways to shutdown a connection. The brute force method is for one side to abort the session by setting the RST flag in the TCP header. The connection is terminated immediately.

The more elegant method takes four steps (Stevens, p.233). In its simplest form, one side (say the server) sends a FIN to indicate it wants to close (Step 35). The client replies with an ACK to that FIN (Step 36) and sends its own FIN to indicate it wants to close its side of the connection too (Step 38). The server then responds with a final ACK. In Figure 10, the classic steps in a shutdown are not followed exactly, with the ACKs of FIN replaced by RST – a hybrid of brute force and elegance. The termination method can vary by application and TCP implementation.

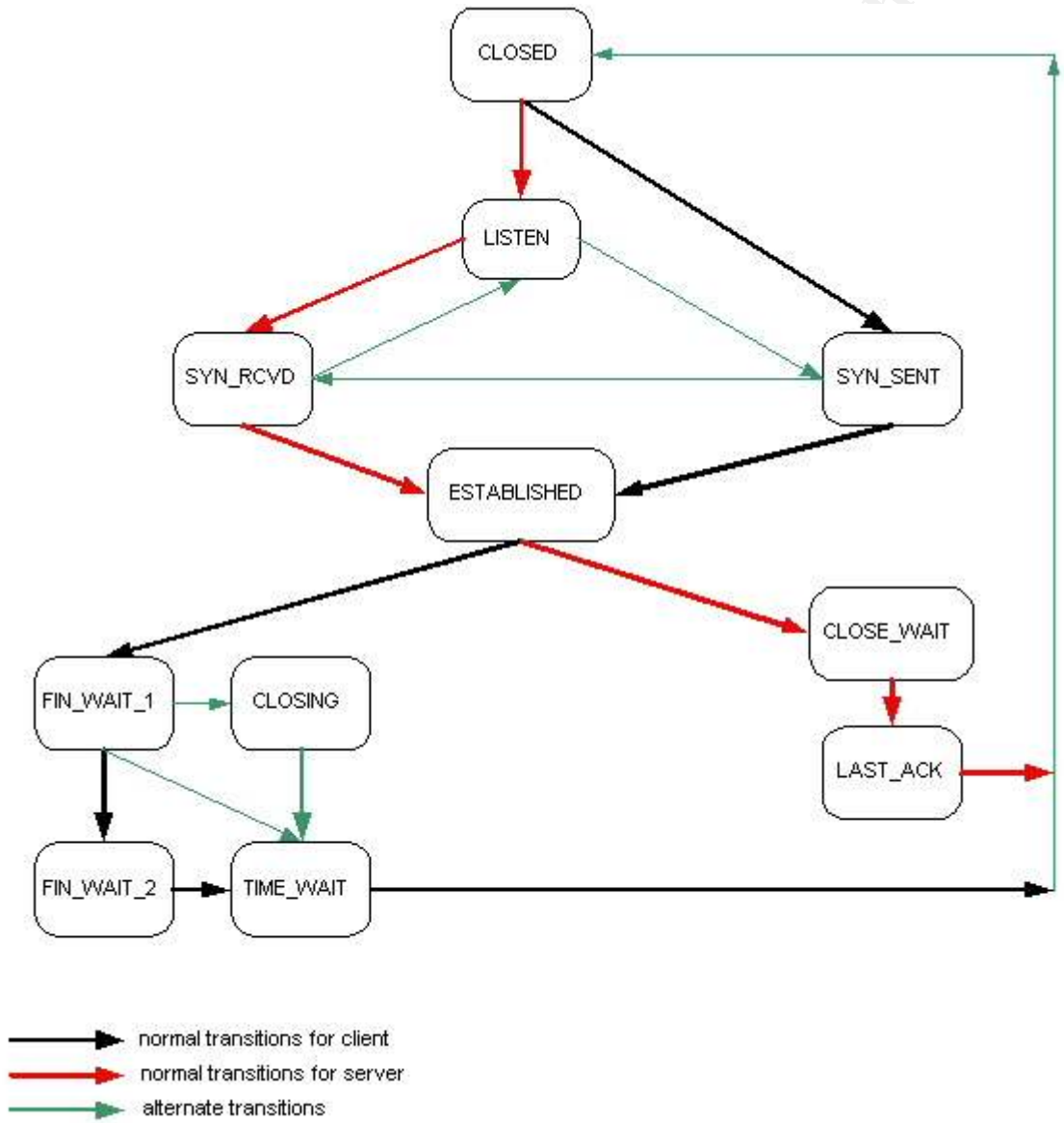
The three connection stages – handshake, data transfer, and shut down – are broad concepts security professionals learn early in their career. But TCP has another higher concept that is not

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

as widely discussed, that of "state". What is the connection doing? Is it in the three-way handshake? Is it established, so data can be sent? It is closed or being closed? Depending on how the client and server applications work, a variety of different situations can occur while a connection is being established or ended, and the TCP protocol handles these conditions by keeping track of the state of both connections.

State

Figure 8 – TCP State Transition Diagram



RFC 793 defines eleven possible states a TCP connection can be in at any given point in the session (ISI-TCP, p. 20). The client and server each have their own state, and when both are "ESTABLISHED", data can be exchanged.

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

Figure 8 illustrates the eleven TCP states, and is based on Figure 18.12 in Stevens (p. 241). It further highlights the fact that TCP connections are full duplex – the client has a connection state (black), the server has it's own connection state (red).

The details of how the connection states change, and what is required for each state change, are described very well in both RFC 793 (sections 3.2, 3.4, 3.5) and in Stevens, chapter 18.6. They are presented here as a head's up to security professionals and/or developers who may one day have to implement TCP/IP in an application.

To see some of these states "in the wild" the netstat command can be used to monitor connections on an SMTP server is running. Figure 9 shows such an example. WITH PERMISSION, open two terminal windows in Unix or Linux. In one window, log on to the same machine as the SMTP server and type "netstat –a -c" to see a list of open ports and their state. The "-c" option runs netstat continuously. On the other terminal, telnet to the SMTP server ("telnet your_SMTP_Server 25"). If the server accepted the connection, netstat will show a state of ESTABLISHED associated with the SMTP server. On another terminal, log onto the SMTP server machine and do a "kill –9" on the process ID of the connection. (Author's note: a "quit" was tried, but then the closing states couldn't be seen – the connection just disappeared from netstat's display.) If the timing is right, netstat may show the connection in FIN_WAIT_1 or FIN_WAIT_2, but most likely it will appear in TIME_WAIT for several iterations before the connection closes. Figure 9 shows an abbreviated example of this exercise. Unfortunately, netstat is not fast enough to catch the three-way handshake.

Figure 9

```
> netstat -ac | grep smtp

tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 devbox-mypc-vm1:32779  devbox-wsrv.devbx:smtp   ESTABLISHED
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
.
.
.
tcp        0      0 devbox-mypc-vm1:32779  devbox-wsrv.devbx:smtp   ESTABLISHED
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
.
.
.
(killed the telnet process using kill -9 on the PID)
tcp        0      0 devbox-mypc-vm1:32779  devbox-wsrv.devbx:smtp   TIME_WAIT
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 devbox-mypc-vm1:32779  devbox-wsrv.devbx:smtp   TIME_WAIT
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 devbox-mypc-vm1:32779  devbox-wsrv.devbx:smtp   TIME_WAIT
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
.
.
.

tcp        0      0 devbox-mypc-vm1:32779  devbox-wsrv.devbx:smtp   TIME_WAIT
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
```

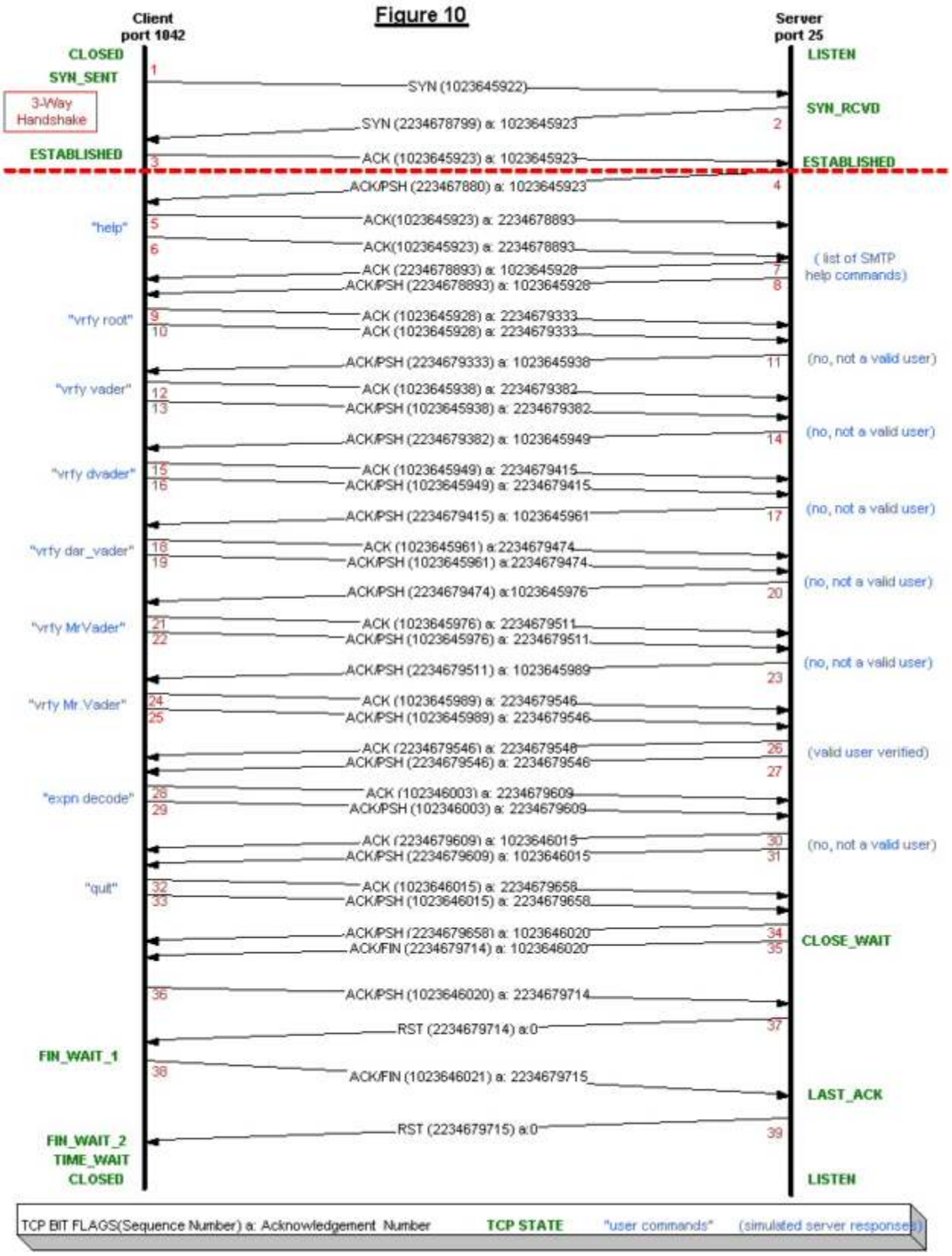
© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

Conclusion

This paper has shown how TCP packets are sent by IP over a network to another computer. Details of how a TCP connection is established, maintained, and terminated have been presented, along with a brief discussion of TCP state. An attempt has been made to delve into areas of TCP connections that are not normally presented in an introductory forum.

To conclude this section, Figure 10 is presented to show an entire TCP session between a client (on port 1042) and an SMTP server, listening on port 25. This session simulates a hacker trying to identify valid users. As denoted by the key at the bottom of the figure, the bit flags of each packet that passes between the client and server are shown (e.g. SYN, ACK, PSH, FIN, RST). The number in parenthesis immediately following the bit flag(s) is the absolute sequence number, and the number following the "a:" is the acknowledgement number. Absolute sequence numbers were used for clarity - although the numbers are large, it more simply demonstrates a connection than relative numbers since the reader does not have to convert back and forth between absolute and relative. Figure 10 is also included here as a resource for security professionals, since a more than two-hour search of the internet trying to find an entire TCP session with absolute sequence numbers came up empty. The commands entered by the client/user are quoted in light blue, and a summary of the server responses are also in light blue and in parentheses.

© SANS Institute 2001, Author retains full rights.



© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

References

Cisco Systems Inc. "Cisco Security Advisory: Cisco IOS Software TCP Initial Sequence Number Randomization Improvements Revision 1.2: Interim". 7 March 2001. http://www.cisco.com/warp/public/707/ios-tcp-isn-random-pub.shtml (20 November 2001).

Davis, Noel. "Predictable Initial Sequence Numbers". 8 May 2001. http://linux.oreillynet.com/pub/a/linux/2001/05/08/insecurities.html (20 November 2001).

Hedrick, C. "RFC 1058 – Routing Information Protocol". June 1988. http://www.ietf.org/rfc/rfc1058.txt?number=1058 (28 November 2001).

Hoelzer, David. "TCP/IP Primer". 2001. http://www.sans.org/NO2001/Hoelzer.pdf (28 November 2001).

(ISI-IP) Information Sciences Institute, University of Southern California. "RFC 791 – Internet Protocol – DARPA Internet Program Protocol Specification". September 1981. http://www.ietf.org/rfc/rfc0791.txt?number=791 (26 November 2001).

(ISI-TCP) Information Sciences Institute, University of Southern California. "RFC 793 – Transmission Control Protocol – DARPA Internet Program Protocol Specification". September 1981. http://www.ietf.org/rfc/rfc0793.txt?number=793 (26 November 2001).

Northcutt, Stephen, Judy Novak and Donald McLachlan. Network Intrusion Detection, An Analyst's Handbook, Second Edition. Indianapolis: New Riders Publishing, 2001.

Rodriguez, Adolpho, John Gatrell, John Karas, and Roland Peschke. "IBM Redbooks – TCP/IP Tutorial and Technical Overview". August 2001. Downloaded PDF file from http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/gg243376.html?Open (20 November, 2001).

Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc., 1994.

© SANS Institute 2001, As part of the Information Security Reading Room. Author retains full rights.

Last Updated: September 1st, 2015

Upcoming SANS Training

Click Here for a full list of all Upcoming SANS Events by Location

SANS Crystal City 2015	Crystal City, VAUS	Sep 08, 2015 - Sep 13, 2015	Live Event
SANS Network Security 2015	Las Vegas, NVUS	Sep 12, 2015 - Sep 21, 2015	Live Event
SANS Seoul 2015	Seoul, KR	Sep 14, 2015 - Sep 19, 2015	Live Event
Data Breach Investigation Summit & Training	Dallas, TXUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Baltimore 2015	Baltimore, MDUS	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Perth 2015	Perth, AU	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS Tallinn 2015	Tallinn, EE	Sep 21, 2015 - Sep 26, 2015	Live Event
SANS ICS Amsterdam 2015	Amsterdam, NL	Sep 22, 2015 - Sep 28, 2015	Live Event
SANS Bangalore 2015	Bangalore, IN	Sep 28, 2015 - Oct 17, 2015	Live Event
SANS DFIR Prague 2015	Prague, CZ	Oct 05, 2015 - Oct 17, 2015	Live Event
SANS Seattle 2015	Seattle, WAUS	Oct 05, 2015 - Oct 10, 2015	Live Event
SANS Tysons Corner 2015	Tysons Corner, VAUS	Oct 12, 2015 - Oct 17, 2015	Live Event
SOS: SANS October Singapore 2015	Singapore, SG	Oct 12, 2015 - Oct 24, 2015	Live Event
SANS Gulf Region 2015	Dubai, AE	Oct 17, 2015 - Oct 29, 2015	Live Event
SANS Cyber Defense San Diego 2015	San Diego, CAUS	Oct 19, 2015 - Oct 24, 2015	Live Event
SANS Tokyo Autumn 2015	Tokyo, JP	Oct 19, 2015 - Oct 31, 2015	Live Event
SANS South Florida 2015	Fort Lauderdale, FLUS	Nov 09, 2015 - Nov 14, 2015	Live Event
SANS Sydney 2015	Sydney, AU	Nov 09, 2015 - Nov 21, 2015	Live Event
SANS London 2015	London, GB	Nov 14, 2015 - Nov 23, 2015	Live Event
Pen Test Hackfest Summit & Training	Alexandria, VAUS	Nov 16, 2015 - Nov 23, 2015	Live Event
SANS Hyderabad 2015	Hyderabad, IN	Nov 24, 2015 - Dec 04, 2015	Live Event
SANS Cape Town 2015	Cape Town, ZA	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS San Francisco 2015	San Francisco, CAUS	Nov 30, 2015 - Dec 05, 2015	Live Event
SANS Milan 2015	OnlineIT	Sep 07, 2015 - Sep 12, 2015	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced